# ECE 530: Cloud Computing

## Homework #1: OpenStack IaaS Deployment

Amber Disher – 101839171 – adisher1@unm.edu

Marshall Hundemer – 101736010 – mhundemer@unm.edu

David Kirby – 101652098 – davidkirby@unm.edu

Spring 2021


# Contents



# List of Figures

## 2 Abstract

OpenStack is an open source cloud computing software designed for automation. For homework #1 we were tasked to deploy OpenStack on a number of virtual machines (VMs) on our system and install some core services. For our purposes, we needed to deploy the following services:

- **Keystone:** Keystone is the identity service used by OpenStack for authentication (authN) and high-level authorization (authZ).
- **Glance:** Glance image services include discovering, registering, and retrieving virtual machine (VM) images. Glance has a RESTful API that allows querying of VM image metadata as well as retrieval of the actual image.
- **Nova:** Compute (Nova) is a cloud computing fabric controller, which is the main part of an IaaS system. It is designed to manage and automate pools of computer resources and can work with widely available virtualization technologies, as well as bare metal and high-performance computing (HPC) configurations.
- **Neutron:** Neutron is an OpenStack project to provide "network connectivity as a service" between interface devices (e.g., vNICs) managed by other OpenStack services (e.g., nova).
- **Horizon:** Dashboard provides administrators and users a graphical interface to access, provision, and automate cloud-based resources. The dashboard is one of several ways users can interact with OpenStack resources.
- **Cinder:** Block Storage provides persistent block-level storage devices for use with OpenStack compute instances. Block storage volumes are fully integrated into OpenStack Compute and the Dashboard allowing for cloud users to manage their own storage needs.

## 3 Introduction

Critically, for this class we were required to deploy these services without the use of automated deployment software (e.g., DevStack) as they would perform all steps and we would learn very little. The instructions we followed were found on the OpenStack website, as well as *Learning OpenStack Networking - Third Edition*, written by James Denton and published by Packt Publishing. We found this book on O'Reilly's book website, to which UNM has graciously provided access for all students and staff.

Once services were deployed, we were then tasked to provision a subnet, a VM, and a storage volume. As we began deploying services, we came to understand that most of these basic services depend on one another to operate and each VM needs to be able to communicate with the others. As such, networking was critical to starting out.

The machine on which we chose to deploy also presented a novel challenge – an Apple silicon-based MacBook Air M1. While this computer satisfied all of the hardware requirements (i.e., 6 core CPU, 6 GB RAM, 200 GB storage), the fact that it was an ARM processor meant that we were limited on which virtualization software to run our VMs. As of this assignment, the authors are only aware of one solution for running native Linux distributions on Apple silicon– a beta technical preview edition of Parallels Desktop for Mac with Apple M1 chip.


Figure 1: Apple Silicon-Based MacBook Air Specs.

# 4 Deployment

For our deployment, we were limited in choices of robust, ARM-based Linux distributions, ultimately choosing Ubuntu Server for ARM, a Ubuntu 20.04.2 LTS edition. We installed a fresh VM using Parallels and Ubuntu Server, making sure to plan for the other nodes required in later steps. Specifically, this required configuring the VMs with three network interfaces – one for outside internet communication and two for internal communication between VMs. It was important to configure the outward-facing interface to share the hosts internet, while the other two interfaces where configured in bridge mode. Our network configurations can be seen in Figure 2, page 4.

For this assignment we needed a minimum of five VMs to accommodate the all of the various services: controller node (`controller01`), compute node (`compute01`), block storage node (`block01`), and two object storage nodes for containers (`object01` & `object02`). To simplify the process, once `controller01` was created, updated, and configured with basic settings, we created linked clones of the VM. This maintained all of the installed packages and settings and saved us from having to start from scratch with each new node. For transparency, below are the commands we ran after initial Ubuntu install. Most of these are personal preferences (e.g., using zsh instead of bash and installing a GUI on top of the server CLI).

```
sudo apt-get install zsh
chsh -s $(which zsh) && sudo chsh -s $(which zsh)
sudo apt update && sudo apt upgrade -y
sudo apt install tasksel -y
sudo tasksel install ubuntu-desktop
sudo systemctl set-default multi-user.target
hostnamectl set-hostname controller01
```

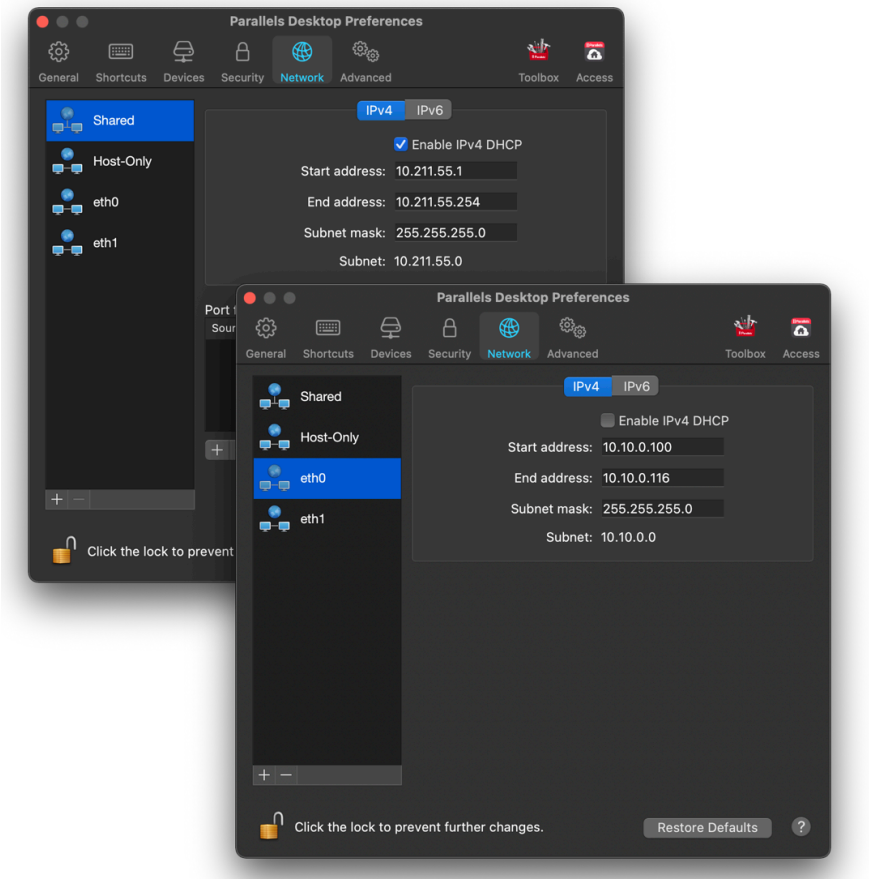


Figure 2: Network Interface Configurations.

Next, we began installing the basic packages needed to deploy OpenStack and would be required on all nodes. It is important to note notation – # denotes command was run as root user, if no prompt is shown, a regular user account was used. Here are the commands used to enable the repository for Ubuntu Cloud Archive, install the python OpenStack client, and MariaDB packages.

```
# add-apt-repository cloud-archive:victoria
# apt install python3-openstackclient
# apt install mariadb-server python3-pymysql
```

Next we configured MariaDB by modifying `/etc/mysql/mariadb.conf.d/99-openstack.cnf` with our `controller01` management IP.

```
[mysqld]

bind-address = 10.211.55.16

default-storage-engine = innodb
innodb_file_per_table = on
max_connections = 4096
collation-server = utf8_general_ci
character-set-server = utf8
```

We restarted and secured MariaDB.

```
# service mysql restart
# mysql_secure_installation
```

Next, we installed the message queue for Ubuntu (rabbitmq); this coordinates operations and status information among services. Note: we used the same password for all services, databases, and nodes with the obvious stipulation that this is a proof-of-concept deployment.

```
# apt install rabbitmq-server
# rabbitmqctl add_user openstack RABBIT_PASS
# rabbitmqctl set_permissions openstack ".*" ".*" ".*"
```

We then installed memcached for Ubuntu allowing for cached tokens and edited the configuration file `/etc/memcached.conf` with our management IP, afterwhich we restarted the memcached service. Note that the OpenStack website instructions called for `python-memcache`, but due to our ARM configuration, we needed to use `python3-memcache`.

```
# apt install memcached python3-memcache
```

```
-l 10.211.55.16
```

```
# service memcached restart
```

The OpenStack documentation gave an option for installing Etcd, a distributed reliable key-value store for distributed key locking, storing configuration, keeping track of service live-ness and other scenarios; however, due to compatibility issues with our python3 setup, we were unable to get Etcd working.

Once our `controller01` node was initialized, we made linked clones of our `controller01` VM to use for our other nodes.

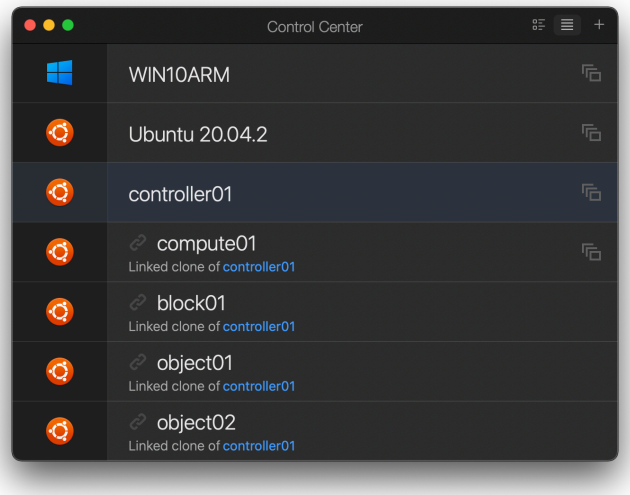


Figure 3: Multiple Nodes Used for OpenStack Deployment.

Knowing how critical it was for our nodes to be able to communicate with each other we went through a series of ping tests to confirm everything was working as expected. We also mapped our managment IPs to their hostnames to make things easier.

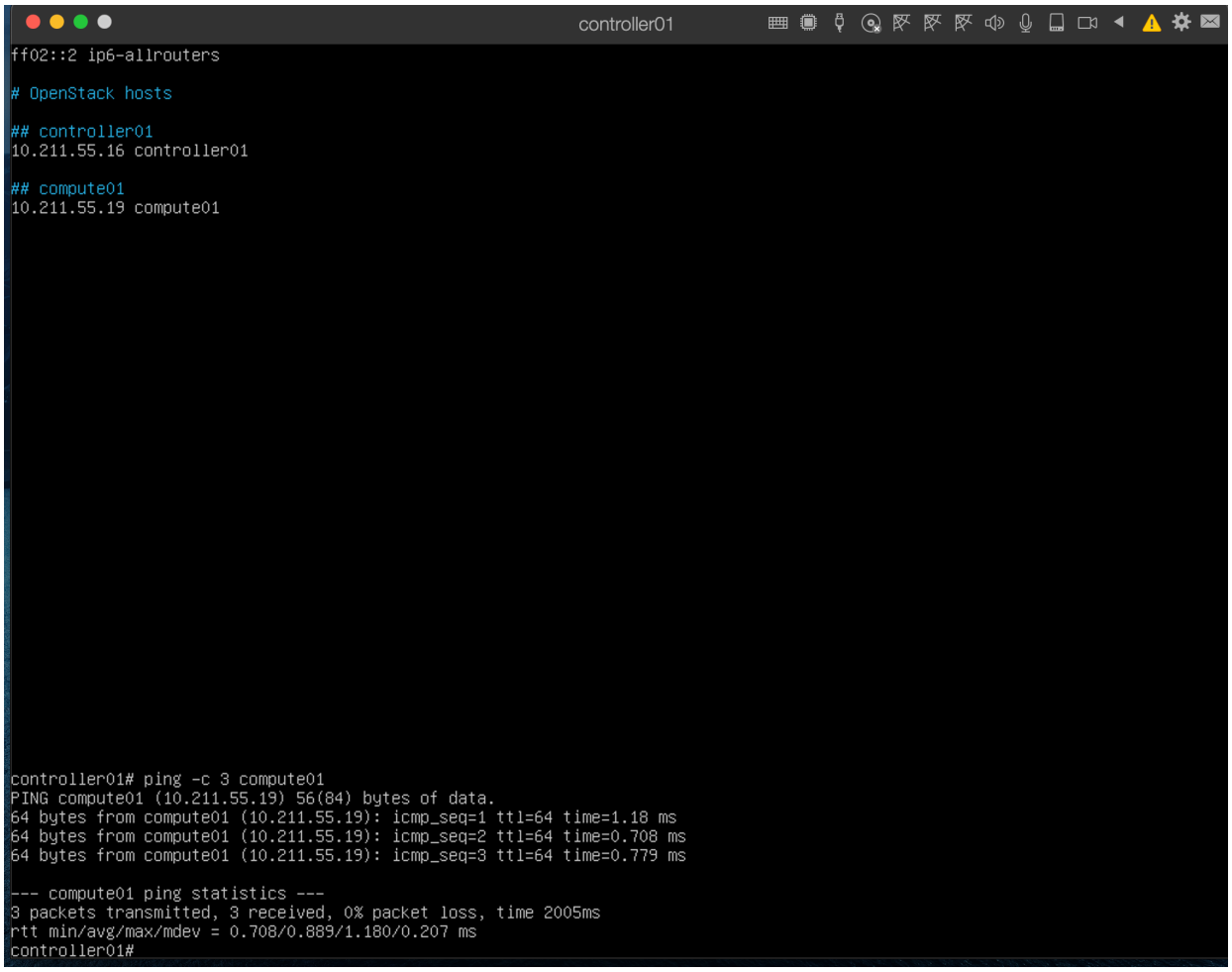


Figure 4: Ping Test Between Nodes.

## 4.1 Keystone

Next, we began deployment of the OpenStack services, starting with the identity service Keystone on the `controller01` node. We started by creating and configuring the identity service in MariaDB.

```
# mysql
MariaDB [(none)]> CREATE DATABASE keystone;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO
    'keystone'@'localhost' IDENTIFIED BY 'KEYSTONE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON keystone.* TO
    'keystone'@'%' IDENTIFIED BY 'KEYSTONE_DBPASS';
exit;
```

Once MariaDB was configured, we set out installing and configuring Keystone.

```
# apt install keystone
```

And editing the config file `/etc/keystone/keystone.conf`

```
[database]
# ...
connection = mysql+pymysql://keystone:KEYSTONE_DBPASS@controller/
    keystone
```

Populating the database.

```
# su -s /bin/sh -c "keystone-manage db_sync" keystone
```

Initializing the Fernet key repositories which allows Keystone to run keystone under another operating system user/group.

```
# keystone-manage fernet_setup --keystone-user keystone
    --keystone-group keystone
# keystone-manage credential_setup --keystone-user keystone
    --keystone-group keystone
```

Next we needed to bootstrap the Keystone service. Note that while OpenStack website calls there controller node `controller`, we called ours `controller01`. We followed the example from *Learning OpenStack Networking* and allowed for expansion to multiple controller nodes if necessary. We use this same naming convention for the other nodes, which proved beneficial when we introduced multiple object nodes later in the deployment.

```
# keystone-manage bootstrap --bootstrap-password ADMIN_PASS
    --bootstrap-admin-url http://controller01:5000/v3/
    --bootstrap-internal-url http://controller01:5000/v3/
    --bootstrap-public-url http://controller01:5000/v3/
    --bootstrap-region-id RegionOne
```

Once keystone was bootstrapped, we restarted our Apache server and began configuring our first keystone accounts (`admin` and `demo`) as well as some projects and roles. We then assigned these users to our newly created roles.

```
# service apache2 restart
```

```
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_PROJECT_NAME=admin
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_DOMAIN_NAME=Default
export OS_AUTH_URL=http://controller01:5000/v3
export OS_IDENTITY_API_VERSION=3
```

```
openstack domain create --description "An Example Domain" example
openstack project create --domain default --description
    "Service Project" service
openstack project create --domain default --description
    "Demo Project" myproject
openstack user create --domain default --password-prompt demo
openstack role create myrole
openstack role add --project myproject --user demo myrole
```

We then needed to test our Keystone installation by unsetting the login credentials and requesting an `admin` authenication token and then a `demo` authenication token. **This satisfies the grading checklist for Keystone (see Figure 5, page 8, Figure 6, page 9, and Figure 7, page 9).**

```
unset OS_AUTH_URL OS_PASSWORD
openstack --os-auth-url http://controller01:5000/v3
    --os-project-domain-name Default
    --os-user-domain-name Default
    --os-project-name admin
    --os-username admin token issue
openstack --os-auth-url http://controller01:5000/v3
    --os-project-domain-name Default
    --os-user-domain-name Default
    --os-project-name myproject
    --os-username demo token issue
```

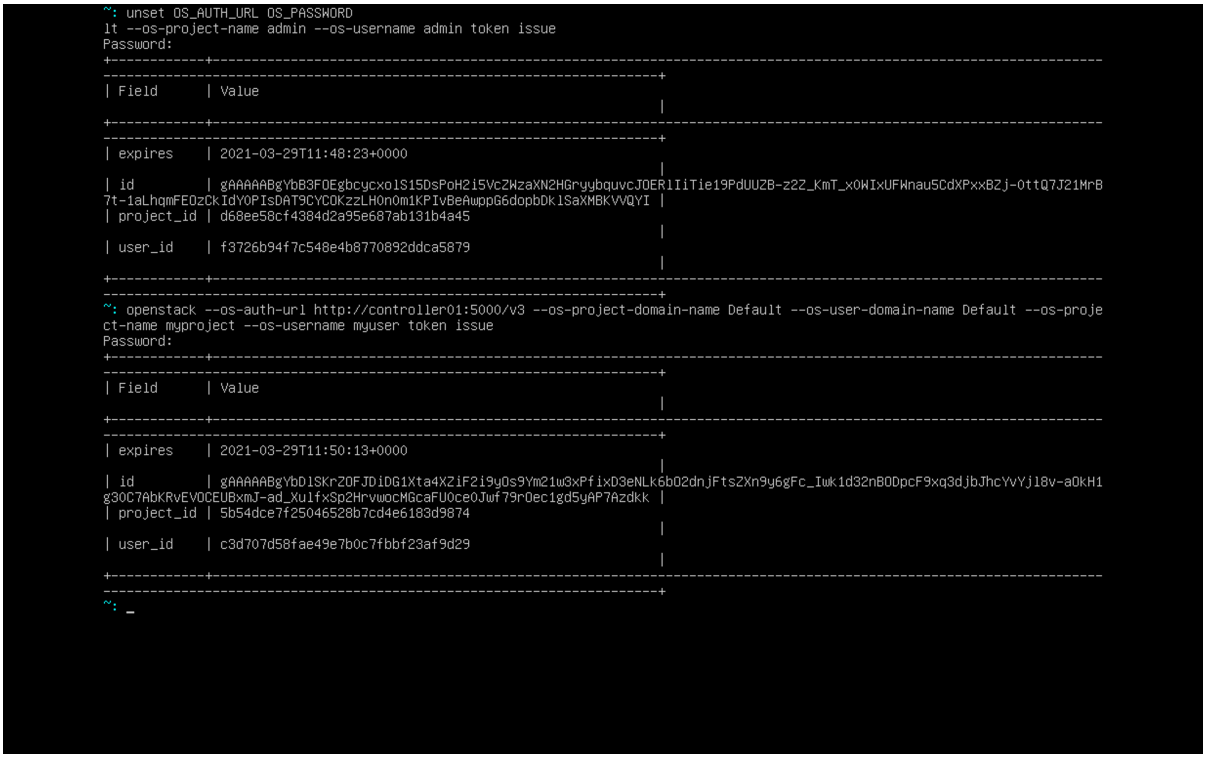


Figure 5: Keystone: Generate Token.

To make things easier for deploying the remaining services, OpenStack recommended creating OpenStack client environment scripts for the `admin` `demo`users, respectively. Using the scripts allows us

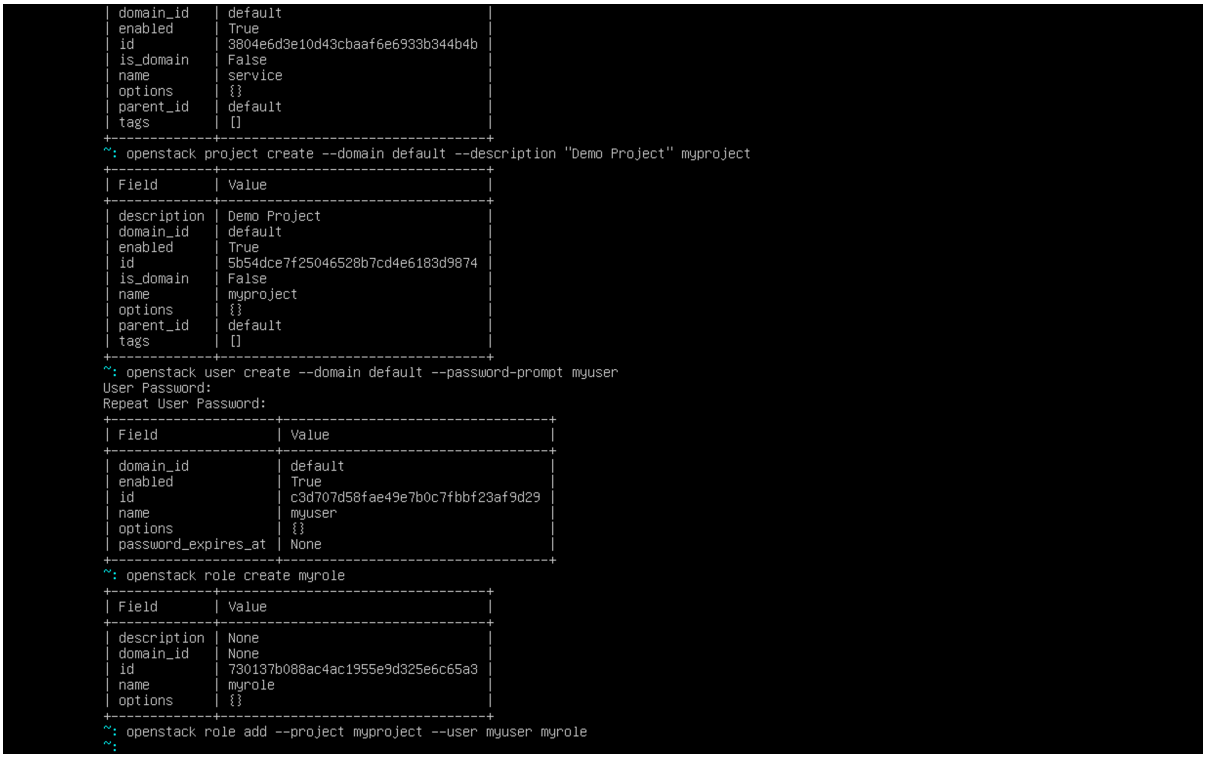


Figure 6: Keystone: Create Users.

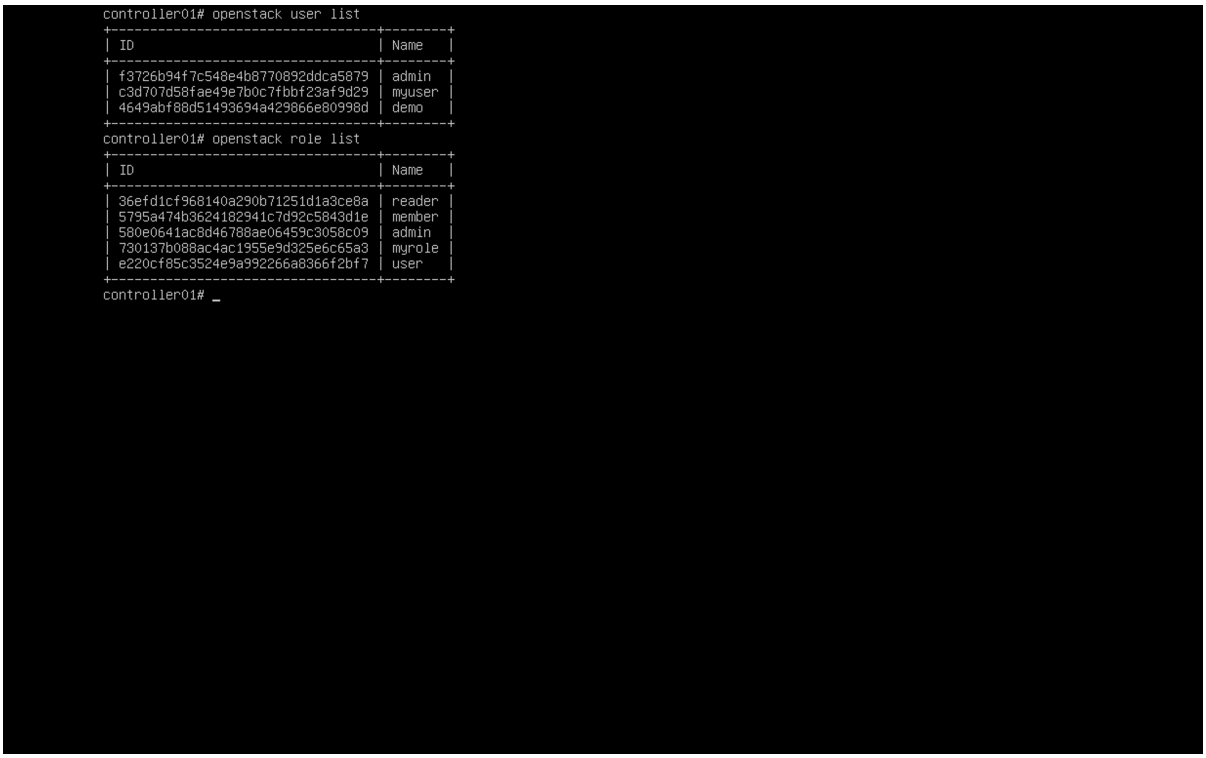


Figure 7: Keystone: Retrieve Users & Roles.

to source admin authenication privileges by entering `source admin-openrc`.

```
export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=admin
export OS_USERNAME=admin
export OS_PASSWORD=ADMIN_PASS
export OS_AUTH_URL=http://controller01:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2

export OS_PROJECT_DOMAIN_NAME=Default
export OS_USER_DOMAIN_NAME=Default
export OS_PROJECT_NAME=myproject
export OS_USERNAME=myuser
export OS_PASSWORD=DEMO_PASS
export OS_AUTH_URL=http://controller01:5000/v3
export OS_IDENTITY_API_VERSION=3
export OS_IMAGE_API_VERSION=2
```

For example, we are able to issue a token as shown in Figure 8, page 10.

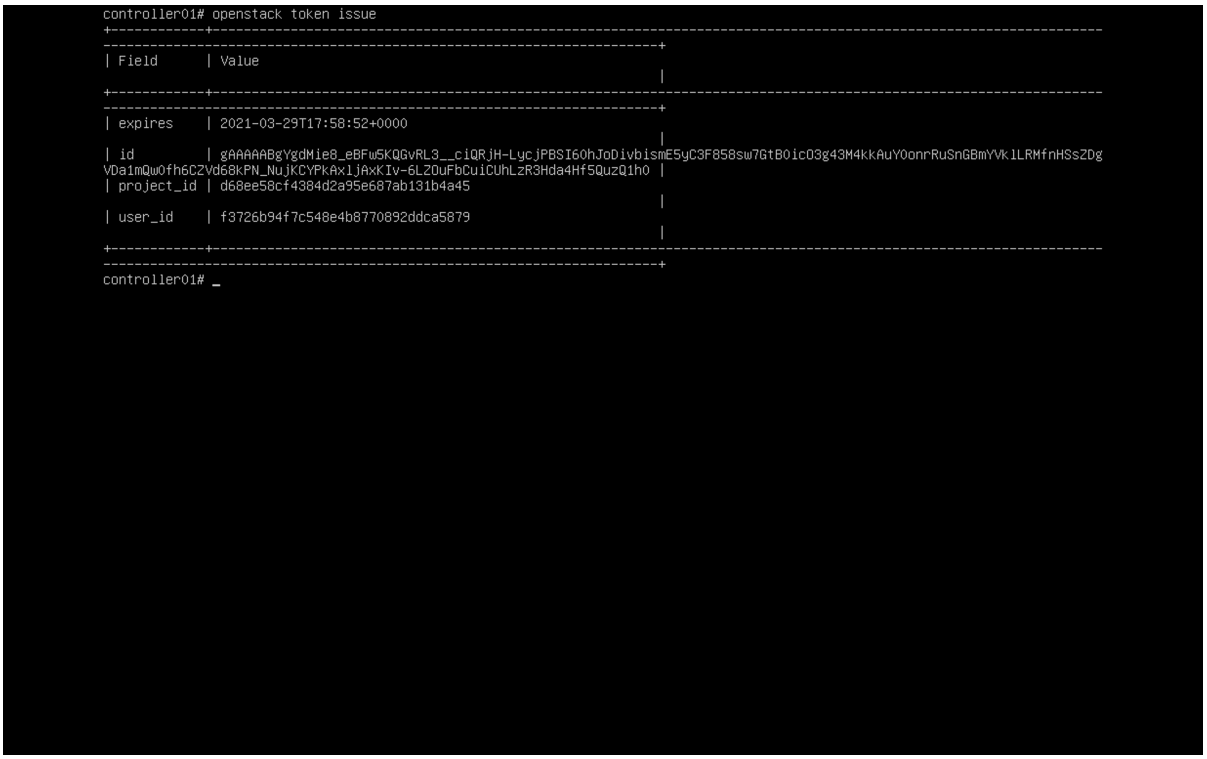


Figure 8: Keystone: User Scripts.

## 4.2 Glance

Once Keystone was installed, we began installing the remaining services. Glance is OpenStack's image services. Because of our ARM configuration, we required using an image file that was also compatible with ARM. Since our homework guidelines called for Cirros, we opted for the latest ARM version – 0.5.2-aarch64. First, we needed to setup the database for Glance.

```
# mysql
MariaDB [(none)]> CREATE DATABASE glance;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO
    'glance'@'localhost' IDENTIFIED BY 'GLANCE_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON glance.* TO
    'glance'@'%' IDENTIFIED BY 'GLANCE_DBPASS';
exit;
```

Then creating our Glance user with the newly created Keystone tools.

```
openstack user create --domain default --password-prompt glance
openstack role add --project service --user glance admin
openstack service create --name glance --description
    "OpenStack Image" image
openstack endpoint create --region RegionOne image public
    http://controller01:9292
openstack endpoint create --region RegionOne image internal
    http://controller01:9292
openstack endpoint create --region RegionOne image admin
    http://controller01:9292
```

Once the database and identity services were set up, we began installing and configuring Glance at `/etc/glance/glance-api.conf`

```
# apt install glance
```

```
[database]
connection =
    mysql+pymysql://glance:GLANCE_DBPASS@controller01/glance

[keystone_authtoken]
www_authenticate_uri = http://controller01:5000
auth_url = http://controller01:5000
memcached_servers = controller01:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = glance
password = GLANCE_PASS
```

```
[paste_deploy]
flavor = keystone

[glance_store]
stores = file,http
default_store = file
filesystem_store_datadir = /var/lib/glance/images/
```

Populate the image service database, restart the service, and verify operation.

```
# su -s /bin/sh -c "glance-manage db_sync" glance
# service glance-api restart

source admin-openrc
wget https://github.com/cirros-dev/cirros/releases/download/0.5.2/
    cirros-0.5.2-aarch64-disk.img
glance image-create --name "cirros" --file
    cirros-0.5.2-aarch64-disk.img --disk-format qcow2
    --container-format bare --visibility=public

glance image-list
```

By importing Cirros and retrieving the images list, **this satisfies the grading checklist for Glance (see Figure 9, page 12 and Figure 10, page 13)**

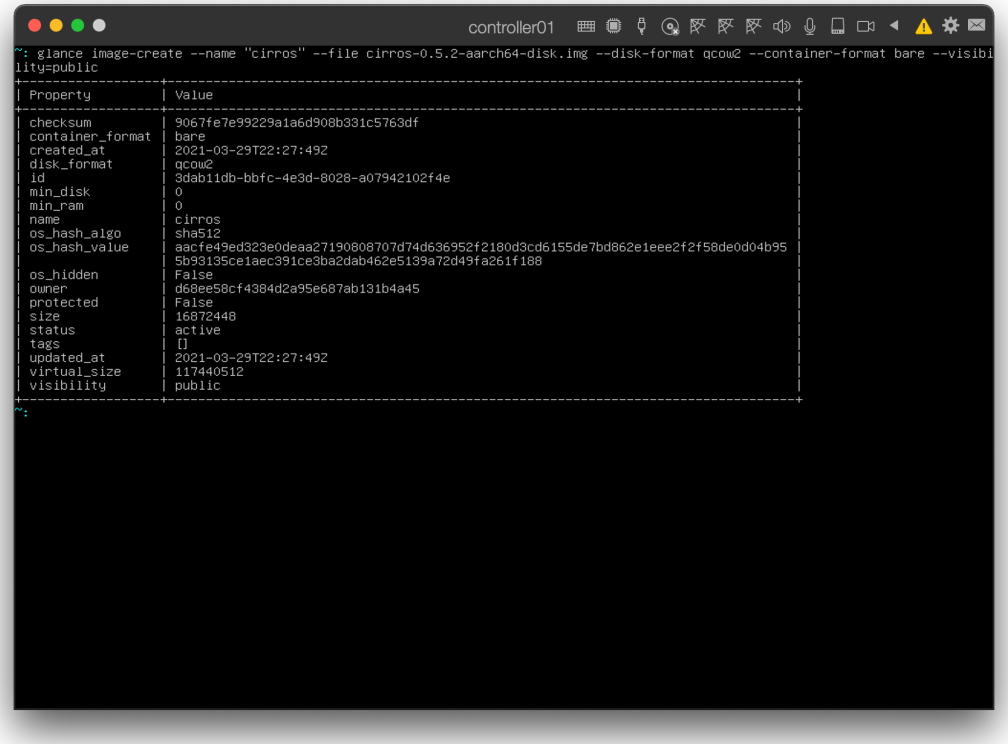


Figure 9: Glance: Import Cirros.

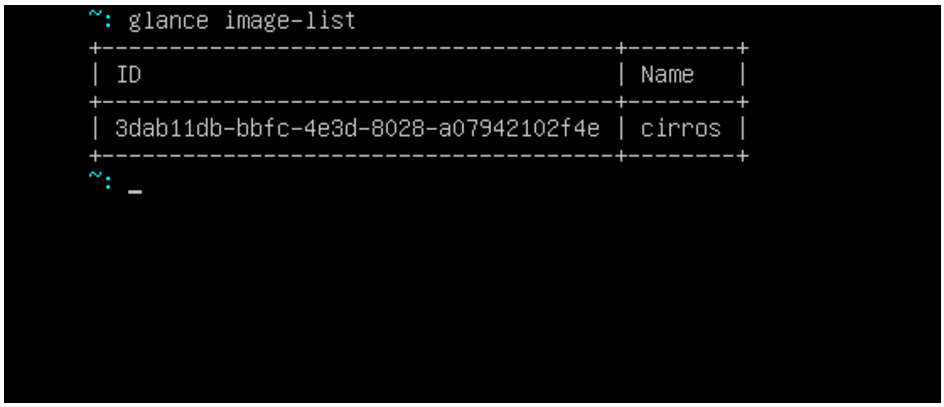


Figure 10: Glance: Image List.

## 4.3 Nova

Once we had a working instance of Keystone and Glance, we moved on to one of the more complex services, Nova. Most of the early steps are similar for each service – create databases, create service credentials, and create endpoints.

```
# mysql
MariaDB [(none)]> CREATE DATABASE nova_api;
MariaDB [(none)]> CREATE DATABASE nova;
MariaDB [(none)]> CREATE DATABASE nova_cell0;

MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO
    'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_api.* TO
    'nova'@'%' IDENTIFIED BY 'NOVA_DBPASS';

MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO
    'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova.* TO
    'nova'@'%' IDENTIFIED BY 'NOVA_DBPASS';

MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO
    'nova'@'localhost' IDENTIFIED BY 'NOVA_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON nova_cell0.* TO
    'nova'@'%' IDENTIFIED BY 'NOVA_DBPASS';
exit;
```

```
source admin-openrc
openstack user create --domain default --password-prompt nova
openstack role add --project service --user nova admin
openstack service create --name nova --description
    "OpenStack Compute" compute
```

```
openstack endpoint create --region RegionOne compute public
    http://controller01:8774/v2.1
openstack endpoint create --region RegionOne compute internal
    http://controller01:8774/v2.1
openstack endpoint create --region RegionOne compute admin
    http://controller01:8774/v2.1
```

We also needed to install the Placement service in support of Nova.

```
# mysql
MariaDB [(none)]> CREATE DATABASE placement;

MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO
    'placement'@'localhost' IDENTIFIED BY 'PLACEMENT_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON placement.* TO
    'placement'@'%' IDENTIFIED BY 'PLACEMENT_DBPASS';
exit;
```

```
source admin-openrc
openstack user create --domain default --password-prompt placement
openstack role add --project service --user placement admin
openstack service create --name placement --description
    "Placement API" placement
openstack endpoint create --region RegionOne placement
    public http://controller01:8778
openstack endpoint create --region RegionOne placement
    internal http://controller01:8778
openstack endpoint create --region RegionOne placement
    admin http://controller01:8778
```

Once Placement and Nova databases were setup, we began installing and configuring the services.

```
# apt install placement-api
```

Configured `/etc/placement/placement.conf`

```
[placement_database]
connection = mysql+pymysql://placement:PLACEMENT_DBPASS@
    controller01/placement
[api]
auth_strategy = keystone

[keystone_authtoken]
auth_url = http://controller01:5000/v3
memcached_servers = controller01:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = placement
password = PLACEMENT_PASS
```

Configuring and verifying Placement service.

```
# su -s /bin/sh -c "placement-manage db sync" placement
# service apache2 restart
source admin-openrc
placement-status upgrade check
pip3 install osc-placement
openstack --os-placement-api-version 1.2 resource class list
    --sort-column name
openstack --os-placement-api-version 1.6 trait list
    --sort-column name
```

```
# apt install nova-api nova-conductor nova-novncproxy
    nova-scheduler
```

Configured `/etc/nova/nova.conf`

```
[api_database]
connection = mysql+pymysql://nova:NOVA_DBPASS@
    controller01/nova_api

[database]
connection = mysql+pymysql://nova:NOVA_DBPASS@
    controller01/nova

[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@
    controller01:5672/
my_ip = 10.211.55.16

[api]
auth_strategy = keystone
```

```
[keystone_authtoken]
www_authenticate_uri = http://controller01:5000/
auth_url = http://controller01:5000/
memcached_servers = controller01:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = NOVA_PASS

[neutron]
auth_url = http://controller01:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET

[vnc]
enabled = true
server_listen = $my_ip
server_proxyclient_address = $my_ip

[glance]
api_servers = http://controller01:9292

[oslo_concurrency]
lock_path = /var/lib/nova/tmp

[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller01:5000/v3
username = placement
password = PLACEMENT_PASS
```

Next, populating the database based on these config files, verifying operation, and restarting services.

```
# su -s /bin/sh -c "nova-manage api_db sync" nova
# su -s /bin/sh -c "nova-manage cell_v2 map_cell0" nova
# su -s /bin/sh -c "nova-manage cell_v2 create_cell
    --name=cell1 --verbose" nova
# su -s /bin/sh -c "nova-manage db sync" nova
# su -s /bin/sh -c "nova-manage cell_v2 list_cells" nova

# service nova-api restart
# service nova-scheduler restart
# service nova-conductor restart
# service nova-novncproxy restart
```

At this point, we have only configured our controller node. The next few steps show how we configured our compute node (`compute01`). Note that we need to use the management IP of the `compute01` node for this configuration.

```
# apt install nova-compute
```

Configure `/etc/nova/nova.conf`.

```
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller01
[api]
auth_strategy = keystone

[keystone_authtoken]
www_authenticate_uri = http://controller01:5000/
auth_url = http://controller01:5000/
memcached_servers = controller01:11211
auth_type = password
project_domain_name = Default
user_domain_name = Default
project_name = service
username = nova
password = NOVA_PASS

[DEFAULT]
my_ip = 10.211.55.19

[neutron]
auth_url = http://controller01:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
```

```
[vnc]
enabled = true
server_listen = 0.0.0.0
server_proxyclient_address = $my_ip
novncproxy_base_url = http://controller01:6080/vnc_auto.html

[glance]
api_servers = http://controller01:9292

[oslo_concurrency]
lock_path = /var/lib/nova/tmp

[placement]
region_name = RegionOne
project_domain_name = Default
project_name = service
auth_type = password
user_domain_name = Default
auth_url = http://controller01:5000/v3
username = placement
password = PLACEMENT_PASS
```

To wrap things up with the compute node, we needed to determine whether our compute node supports hardware acceleration for virtual machines. Given that Parallels is still in technical preview, it's no surprise that it did not support hardware acceleration. Therefore, we needed to configure libvirt to use QEMU instead of KVM by editing `/etc/nova/nova-compute.conf`

```
egrep -c '(vmx|svm)' /proc/cpuinfo
```

```
[libvirt]
virt_type = qemu
```

Next we needed to set up and verify Nova on the compute node, Figure 10, page 13 confirms this.

```
# service nova-compute restart
source admin-openrc
openstack compute service list --service nova-compute
# su -s /bin/sh -c "nova-manage cell_v2 discover_hosts
    --verbose" nova
openstack compute service list
openstack catalog list
openstack image list
nova-status upgrade check
```

The VM we created is shown in the conclusion section Figure 18, page 34. **This satisfies the grading checklist for Nova.**

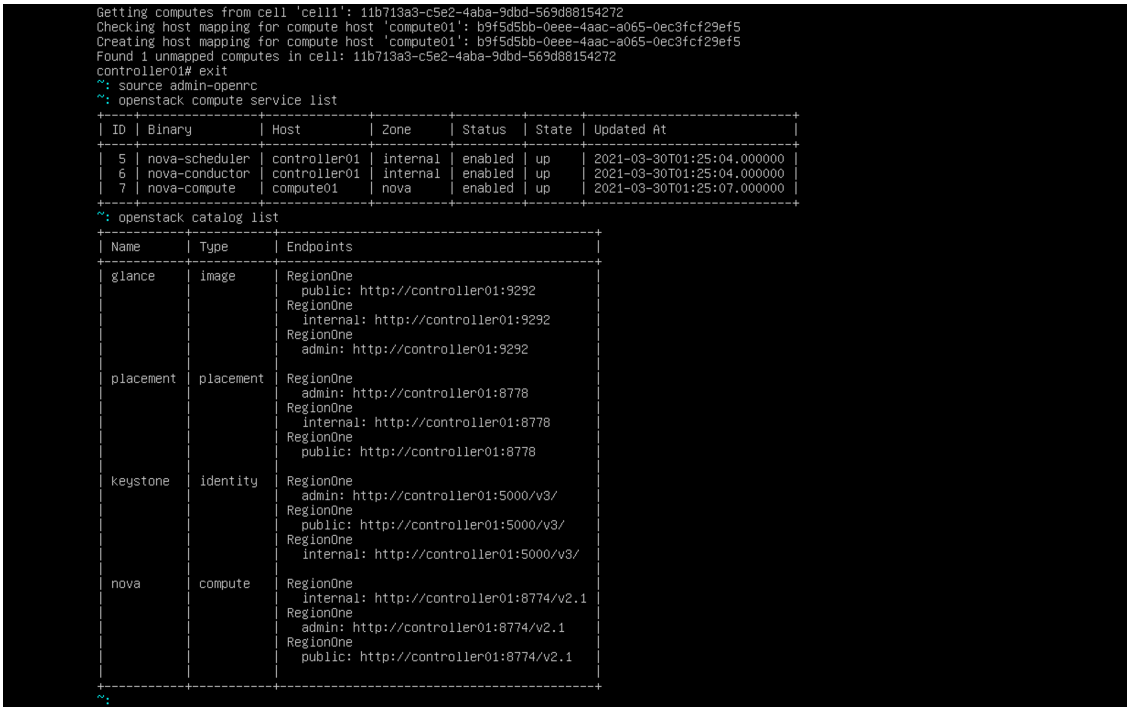


Figure 11: Nova: Catalog List.

## 4.4 Neutron

Before we could create a VM as called for in the homework instructions, we needed to install a networking service. While the homework called for Nova-networking, that service has been deprecated and with approval from the professor opted for installing Neutron instead. First we needed to configure name resolutions. In our case, we did most of this during the initial network setup by configuring `/etc/hosts` on nodes `controller01`, `compute01`, and `block01`.

```
# controller
10.211.55.16        controller01

# compute1
10.211.55.19        compute01

# block1
10.211.55.34        block01

# object1
10.211.55.41        object01

# object2
10.211.55.47        object02
```

After verifying all nodes were able to ping each other and external sites, we proceeded with install Neutron, first on the controller node.

```
# mysql
MariaDB [(none)]> CREATE DATABASE neutron;

MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO
    'neutron'@'localhost' IDENTIFIED BY 'NEUTRON_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON neutron.* TO
    'neutron'@'%' IDENTIFIED BY 'NEUTRON_DBPASS';
exit;
```

```
source admin-openrc
openstack user create --domain default --password-prompt neutron
openstack role add --project service --user neutron admin
openstack service create --name neutron --description
    "OpenStack Networking" network
openstack endpoint create --region RegionOne network public
    http://controller01:9696
openstack endpoint create --region RegionOne network internal
    http://controller01:9696
openstack endpoint create --region RegionOne network admin
    http://controller01:9696
```

We then configured `/etc/neutron/neutron.conf`

```
[DEFAULT]
core_plugin = ml2
service_plugins =
transport_url = rabbit://openstack:RABBIT_PASS@controller01
auth_strategy = keystone
notify_nova_on_port_status_changes = true
notify_nova_on_port_data_changes = true

[keystone_authtoken]
www_authenticate_uri = http://controller01:5000
auth_url = http://controller01:5000
memcached_servers = controller01:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = neutron
password = NEUTRON_PASS

[nova]
auth_url = http://controller01:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = nova
password = NOVA_PASS
```

```
[database]
connection = mysql+pymysql://neutron:NEUTRON_DBPASS@
    controller01/neutron

[oslo_concurrency]
lock_path = /var/lib/neutron/tmp
```

Next we needed to configure `/etc/neutron/plugins/ml2/ml2_conf.ini`, the ML2 plugin which uses the Linux bridge mechanism to build layer-2 (bridging and switching) virtual networking infrastructure for instances.

```
[ml2]
type_drivers = flat,vlan
tenant_network_types =
mechanism_drivers = linuxbridge
extension_drivers = port_security

[ml2_type_flat]
flat_networks = provider

[securitygroup]
enable_ipset = true
```

Then configured the Linux bridge plugin `/etc/neutron/plugins/ml2/linuxbridge_agent.ini`.

```
[linux_bridge]
physical_interface_mappings = provider:eth0

[vxlan]
enable_vxlan = false

[securitygroup]
enable_security_group = true
firewall_driver = neutron.agent.linux.iptables_firewall.
    IptablesFirewallDriver
net.bridge.bridge-nf-call-iptables
net.bridge.bridge-nf-call-ip6tables
```

Finally, we needed to modify `/etc/neutron/dhcp_agent.ini`.

```
[DEFAULT]
interface_driver = linuxbridge
dhcp_driver = neutron.agent.linux.dhcp.Dnsmasq
enable_isolated_metadata = true
```

Once the controller node was configured, we moved to configuring the compute node and the file `/etc/nova/nova.conf`.

```
[neutron]
auth_url = http://controller01:5000
auth_type = password
project_domain_name = default
user_domain_name = default
region_name = RegionOne
project_name = service
username = neutron
password = NEUTRON_PASS
service_metadata_proxy = true
metadata_proxy_shared_secret = METADATA_SECRET
```

To finalize and verify the Neutron service, we ran the following commands on `controller01` and `compute01`, respectively.

```
# ln -s /etc/apparmor.d/usr.sbin.dnsmasq /etc/apparmor.d/disable/
# systemctl restart apparmor
# systemctl restart openstack-nova-api.service
# systemctl enable openstack-neutron.service
    openstack-neutron-linuxbridge-agent.service
    openstack-neutron-dhcp-agent.service
    openstack-neutron-metadata-agent.service
# systemctl start openstack-neutron.service
    openstack-neutron-linuxbridge-agent.service
    openstack-neutron-dhcp-agent.service
    openstack-neutron-metadata-agent.service
```

```
# systemctl restart openstack-nova-compute.service
# systemctl enable openstack-neutron-linuxbridge-agent.service
# systemctl start openstack-neutron-linuxbridge-agent.service
```

To verify everything is working correctly:

```
source admin-openrc
openstack extension list --network
openstack network agent list
```

This finally allows us to create a network and subnet for use with our images, which in this example is named "provider". **This satisfies the grading checklist for Neutron (see Figure 12, page 23).**

```
source admin-openrc
openstack network create --share --external
    --provider-physical-network provider
    --provider-network-type flat provider
openstack subnet create --network provider
    --allocation-pool start=203.0.113.101,end=203.0.113.250
    --dns-nameserver 8.8.4.4 --gateway 203.0.113.1
    --subnet-range 203.0.113.0/24 provider
```

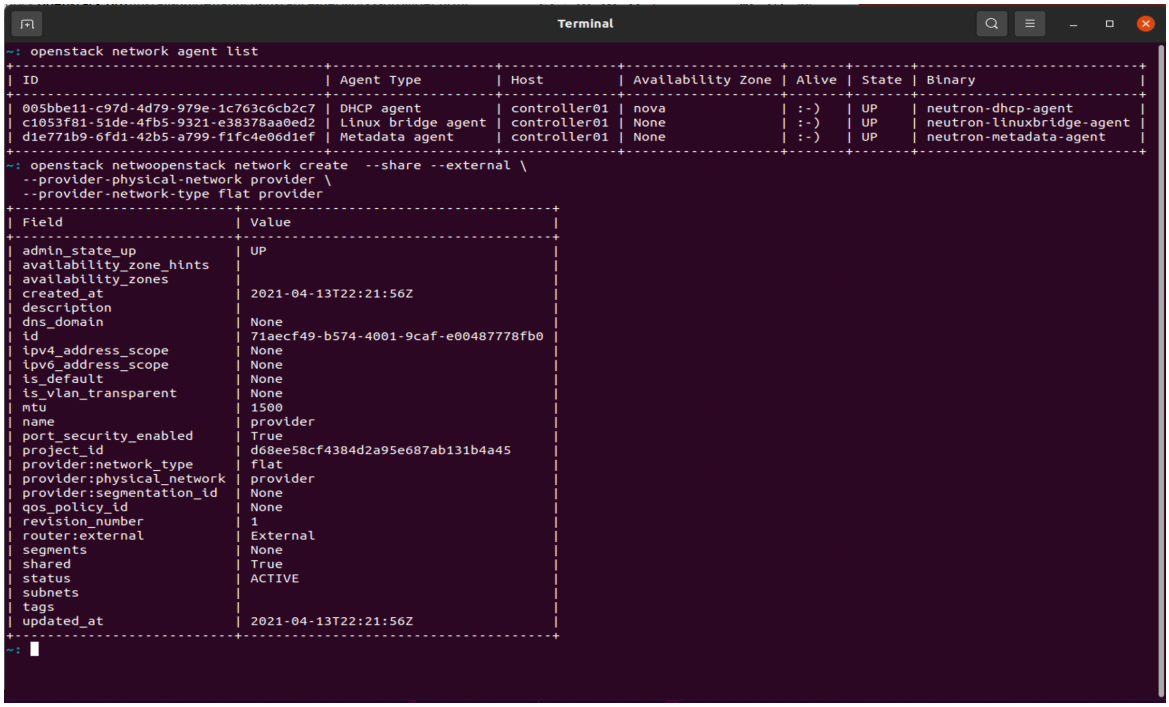


Figure 12: Neutron: Create Network.

## 4.5 Horizon

Thus far, everything we had done was in the CLI, but Horizon gives a graphical dashboard through which we can create instances, manage users, and install new images.

```
# apt install openstack-dashboard
```

Configure `/etc/openstack-dashboard/local_settings.py`.

```
OPENSTACK_HOST = "controller01"
ALLOWED_HOSTS = ['*']
SESSION_ENGINE = 'django.contrib.sessions.backends.cache'

CACHES = {
    'default': {
    'BACKEND':'django.core.cache.backends.memcached.MemcachedCache',
    'LOCATION': 'controller:11211',
    }
}
OPENSTACK_KEYSTONE_URL = "http://%s/identity/v3" % OPENSTACK_HOST
OPENSTACK_KEYSTONE_MULTIDOMAIN_SUPPORT = True
OPENSTACK_API_VERSIONS = {
    "identity": 3,
    "image": 2,
    "volume": 3,
}
OPENSTACK_KEYSTONE_DEFAULT_DOMAIN = "Default"
OPENSTACK_KEYSTONE_DEFAULT_ROLE = "user"
OPENSTACK_NEUTRON_NETWORK = {
    'enable_router': False,
    'enable_quotas': False,
    'enable_ipv6': False,
    'enable_distributed_router': False,
    'enable_ha_router': False,
    'enable_lb': False,
    'enable_firewall': False,
    'enable_vpn': False,
    'enable_fip_topology_check': False,
}
```

Then after modifying `/etc/apache2/conf-available/openstack-dashboard.conf`

```
WSGIApplicationGroup %{GLOBAL}
```

```
# systemctl reload apache2.service
```

We were able to restart our Apache server to finalize installation. To verify installation, we navigate to http://controller01/horizon. This works both inside and outside of our VMs as we have opened it up to the external network. Logging in with the user/password combination we created in Keystone, we are able to see the dashboard. **This satisfies the grading checklist for Horizon (see Figure 13, page 25).**

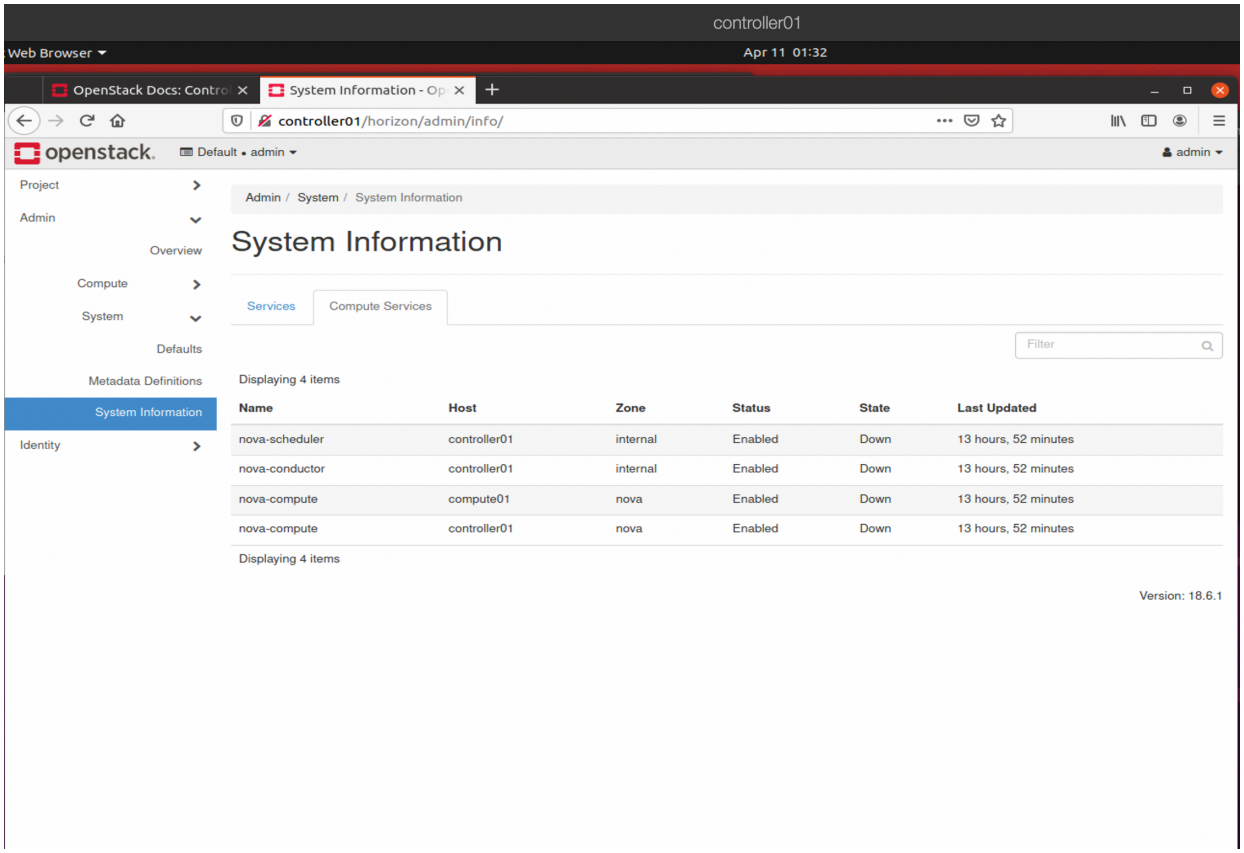


Figure 13: Horizon: Services Information.

## 4.6 Cinder

This section describes how to install and configure storage nodes for the block storage service to be used on `block01` node. As with the other nodes, we need to configure MariaDB and Keystone.

```
# mysql
MariaDB [(none)]> CREATE DATABASE cinder;
MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO
    'cinder'@'localhost' IDENTIFIED BY 'CINDER_DBPASS';
MariaDB [(none)]> GRANT ALL PRIVILEGES ON cinder.* TO
    'cinder'@'%' IDENTIFIED BY 'CINDER_DBPASS';
exit;
```

```
source admin-openrc
openstack user create --domain default --password-prompt cinder
openstack role add --project service --user cinder admin
openstack service create --name cinderv2 --description
    "OpenStack Block Storage" volumev2
openstack service create --name cinderv3 --description
    "OpenStack Block Storage" volumev3
```

```
openstack endpoint create --region RegionOne volumev2
    public http://controller01:8776/v2/%\(project_id\)s
openstack endpoint create --region RegionOne volumev2
    internal http://controller01:8776/v2/%\(project_id\)s
openstack endpoint create --region RegionOne volumev2
    admin http://controller01:8776/v2/%\(project_id\)s
openstack endpoint create --region RegionOne volumev3
    public http://controller01:8776/v3/%\(project_id\)s
openstack endpoint create --region RegionOne volumev3
    internal http://controller01:8776/v3/%\(project_id\)s
openstack endpoint create --region RegionOne volumev3
    admin http://controller01:8776/v3/%\(project_id\)s
```

On our `controller01` node, we install and configured the Cinder service by modifying the config file `/etc/cinder/cinder.conf`.

```
# apt install cinder-api cinder-scheduler
```

```
[database]
connection = mysql+pymysql://cinder:CINDER_DBPASS@
    controller01/cinder
[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller01
auth_strategy = keystone
my_ip = 10.211.55.16

[keystone_authtoken]
www_authenticate_uri = http://controller01:5000
auth_url = http://controller01:5000
memcached_servers = controller01:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS

[oslo_concurrency]
lock_path = /var/lib/cinder/tmp
```

We populated the database with these changes.

```
# su -s /bin/sh -c "cinder-manage db sync" cinder
```

Next we configured the `compute01` node for Cinder by modifying `/etc/nova/nova.conf`.

```
[cinder]
os_region_name = RegionOne
```

Restarted the `compute01` API and block storage services.

```
# service nova-api restart
# service cinder-scheduler restart
# service apache2 restart
```

The next steps were performed on the `block01` node. We install a local volume management tool, created a LVM physical volume, groups, and configured permissions.

```
# apt install lvm2 thin-provisioning-tools
# pvcreate /dev/sdb
# vgcreate cinder-volumes /dev/sdb
```

Edited the `/etc/lvm/lvm.conf`

```
devices {
    filter = [ "a/sdb/", "r/.*/"]
}
```

After preparing the physical volume, it was time to install Cinder on the `block01` node.

```
# apt install cinder-volume
```

And then configure `/etc/cinder/cinder.conf`

```
[database]
connection = mysql+pymysql://cinder:CINDER_DBPASS@
    controller01/cinder

[DEFAULT]
transport_url = rabbit://openstack:RABBIT_PASS@controller01
auth_strategy = keystone
my_ip = 10.211.55.34
enabled_backends = lvm
glance_api_servers = http://controller01:9292

[keystone_authtoken]
www_authenticate_uri = http://controller01:5000
auth_url = http://controller01:5000
memcached_servers = controller01:11211
auth_type = password
project_domain_name = default
user_domain_name = default
project_name = service
username = cinder
password = CINDER_PASS

[oslo_concurrency]
lock_path = /var/lib/cinder/tmp
```

```
[lvm]
volume_driver = cinder.volume.drivers.lvm.LVMVolumeDriver
volume_group = cinder-volumes
target_protocol = iscsi
target_helper = tgtadm
```

Once configured, we restarted all cinder services on `block01`

```
# service tgt restart
# service cinder-volume restart
```

Once the services were restarted, we created a volume using our newly installed Horizon dashboard. **This satisfies the grading checklist for Cinder (see Figure 14, page 28).**

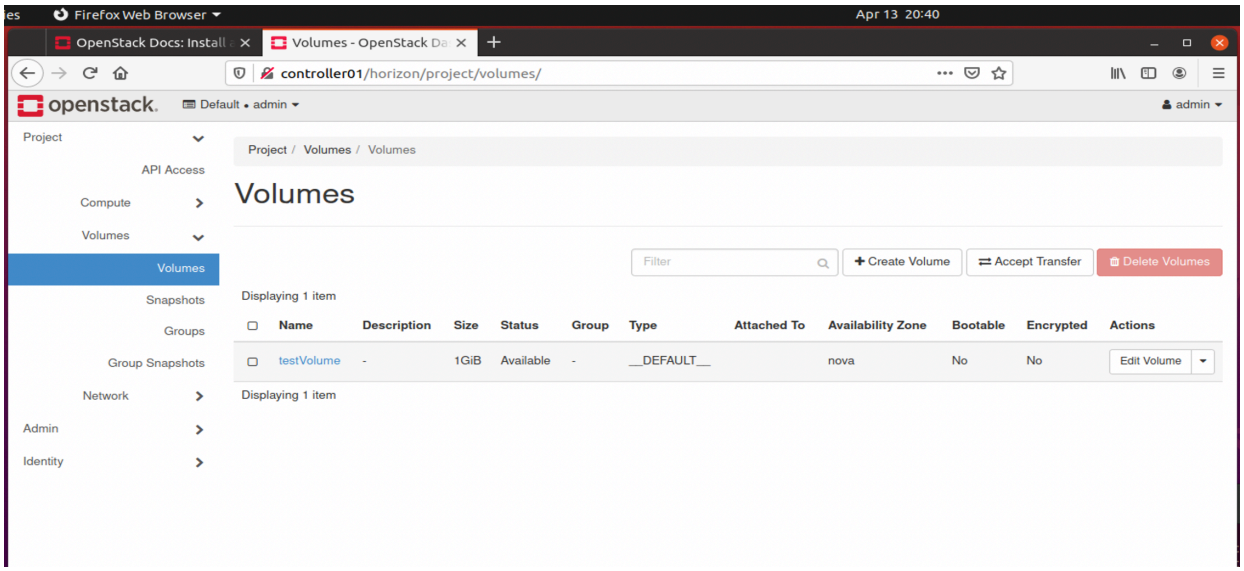


Figure 14: Cinder: Create and List Volumes.

## 4.7 Swift

The last service we were tasked to install was Swift, OpenStack's container service. Running through the same database and Keystone user creation methods as for the other services, we created two more nodes to be used for object storage. Each of these nodes needed to have two physical volumes created as in the block storage node. These multiple nodes and volumes allow us to balance our loads based on demand.

```
source admin-openrc
openstack user create --domain default --password-prompt swift
openstack role add --project service --user swift admin
openstack service create --name swift --description
    "OpenStack Object Storage" object-store
openstack endpoint create --region RegionOne object-store
    public http://controller01:8080/v1/AUTH_%\(project_id\)s
openstack endpoint create --region RegionOne object-store
    internal http://controller01:8080/v1/AUTH_%\(project_id\)s
openstack endpoint create --region RegionOne object-store
    admin http://controller01:8080/v1/AUTH_%\(project_id\)s
# apt-get install swift swift-proxy python-swiftclient
    python-keystoneclient python-keystonemiddleware
# curl -o /etc/swift/proxy-server.conf https://opendev.org/
    openstack/swift/raw/branch/master/etc/proxy-server.conf-sample
```

Edited the config `/etc/swift/proxy-server.conf`

```
[DEFAULT]
bind_port = 8080
user = swift
swift_dir = /etc/swift

[pipeline:main]
pipeline = catch_errors gatekeeper healthcheck proxy-logging
cache container_sync bulk ratelimit authtoken keystoneauth
container-quotas account-quotas slo dlo versioned_writes
proxy-logging proxy-server
[app:proxy-server]
use = egg:swift#proxy
account_autocreate = True
[filter:keystoneauth]
use = egg:swift#keystoneauth
operator_roles = admin,user
[filter:authtoken]
paste.filter_factory = keystonemiddleware.auth_token:
    filter_factory
www_authenticate_uri = http://controller01:5000
auth_url = http://controller01:5000
memcached_servers = controller01:11211
auth_type = password
project_domain_id = default
user_domain_id = default
project_name = service
username = swift
password = SWIFT_PASS
delay_auth_decision = True

[filter:cache]
use = egg:swift#memcache
memcache_servers = controller01:11211
```

This completes the controller node setup, which then led us to setting up the separate object storage

nodes. Since each of these would be identical, we created and configured one, then cloned it, but changed the hostname and updated all config file with corresponding IP addresses.

```
# apt-get install xfsprogs rsync
# mkfs.xfs /dev/sdb
# mkfs.xfs /dev/sdc
# mkdir -p /srv/node/sdb
# mkdir -p /srv/node/sdc

# blkid
UUID="<UUID-from-output-above>" /srv/node/sdb xfs noatime 0 2
UUID="<UUID-from-output-above>" /srv/node/sdc xfs noatime 0 2

# mount /srv/node/sdb
# mount /srv/node/sdc
```

Created the `/etc/rsyncd.conf` and added the following, making sure to adjust the IP based on which object node we were working.

```
uid = swift
gid = swift
log file = /var/log/rsyncd.log
pid file = /var/run/rsyncd.pid
address = 10.211.55.41 (47)

[account]
max connections = 2
path = /srv/node/
read only = False
lock file = /var/lock/account.lock

[container]
max connections = 2
path = /srv/node/
read only = False
lock file = /var/lock/container.lock

[object]
max connections = 2
path = /srv/node/
read only = False
lock file = /var/lock/object.lock
```

We edited the `/etc/default/rsync` file.

```
RSYNC_ENABLE=true
```

Then started the rsync service and began installing services.

```
# service rsync start
# apt-get install swift swift-account swift-container
    swift-object
# curl -o /etc/swift/account-server.conf https://opendev.org/
  openstack/swift/raw/branch/master/etc/account-server.conf-sample
# curl -o /etc/swift/container-server.conf https://opendev.org/
 openstack/swift/raw/branch/master/etc/container-server.conf-sample
# curl -o /etc/swift/object-server.conf https://opendev.org/
    openstack/swift/raw/branch/master/etc/object-server.conf-sample
```

Edited the `/etc/swift/account-server.conf`, `/etc/swift/container-server.conf`, and `/etc/swift/object-server.conf` files with the following data, substituting their respective servers under the pipeline and respective IP addresses based on object node.

```
[DEFAULT]
bind_ip = 10.211.55.41 (47)
bind_port = 6202
user = swift
swift_dir = /etc/swift
devices = /srv/node
mount_check = True

[pipeline:main]
pipeline = healthcheck recon account-server
```

We changed ownership of the mount points to swift.

```
# chown -R swift:swift /srv/node
# mkdir -p /var/cache/swift
# chown -R root:swift /var/cache/swift
# chmod -R 775 /var/cache/swift
```

Once nodes were configured properly, we began creating and distributing initial account, container, and object rings. The ring builder creates configuration files that each node uses to determine and deploy the storage architecture. We need to run the ring builder for each physical volume on each object storage node – a total of four times. We then verified the ring contents and rebalanced the ring. These steps needed to be repeated on the object server, container server, and account server.

```
# cd /etc/swift
# swift-ring-builder account.builder create 10 3 1
# swift-ring-builder account.builder add --region 1 --zone 1
    --ip 10.211.55.41 --port 6202 --device sdb --weight 100
# swift-ring-builder account.builder add --region 1 --zone 1
    --ip 10.211.55.41 --port 6202 --device sdc --weight 100
# swift-ring-builder account.builder add --region 1 --zone 1
    --ip 10.211.55.47 --port 6202 --device sdb --weight 100
# swift-ring-builder account.builder add --region 1 --zone 1
    --ip 10.211.55.47 --port 6202 --device sdc --weight 100

# swift-ring-builder account.builder

# swift-ring-builder account.builder rebalance
```

Figure 15, page 32 shows the output of the rebalancing.

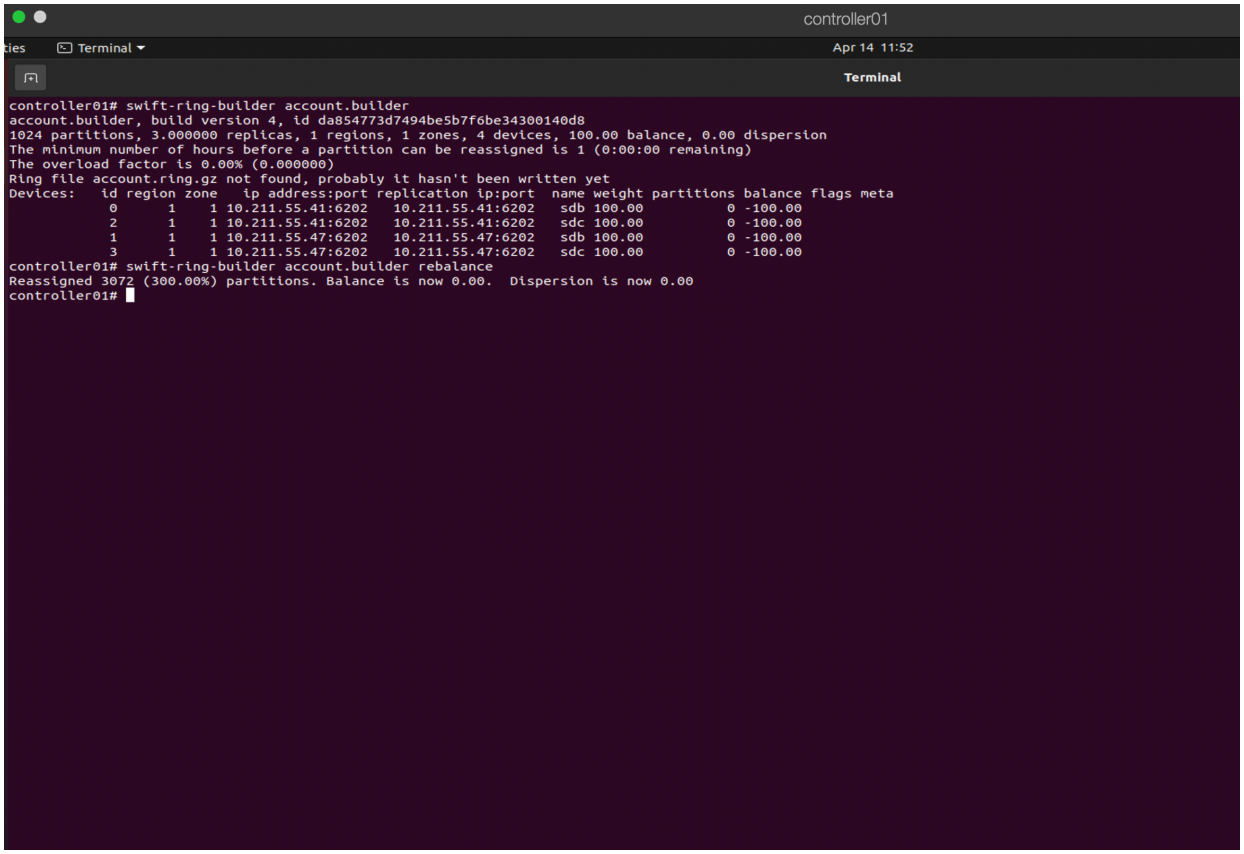


Figure 15: Swift: Rebalancing the Rings.

After distribute the ring configuration files `account.ring.gz`, `container.ring.gz`, and `object.ring.gz` to the `/etc/swift` directory on each storage node, we set out finalizing installation.

We obtained and edited the `/etc/swift/swift.conf` file from the object storage source repository. Once edited as shown below, we copied this file to each of the storage nodes.

```
# curl -o /etc/swift/swift.conf https://opendev.org/
    openstack/swift/raw/branch/master/etc/swift.conf-sample
```

```
[swift-hash]
    swift_hash_path_suffix = HASH_PATH_SUFFIX
    swift_hash_path_prefix = HASH_PATH_PREFIX

[storage-policy:0]
name = Policy-0
default = yes
```

On each node, it was important to give proper permission to this file, then restarted all storage services.

```
# chown -R root:swift /etc/swift
# service memcached restart
# service swift-proxy restart
# swift-init all start
```

Unfortunately, after restarting the nodes, both object nodes became corrupted and could no longer boot into an operating system. Therefore we were unable to create a container or upload/download a file as called for in the homework. We will continue to work on rebuilding the VMs, but it will not be done in time for the deadline.

## 4.8 Heat (extra credit)

We were able to install an extra orchestration service, Heat. Figures 16, page 33 show our Horizon dashboard with the Heat services running.

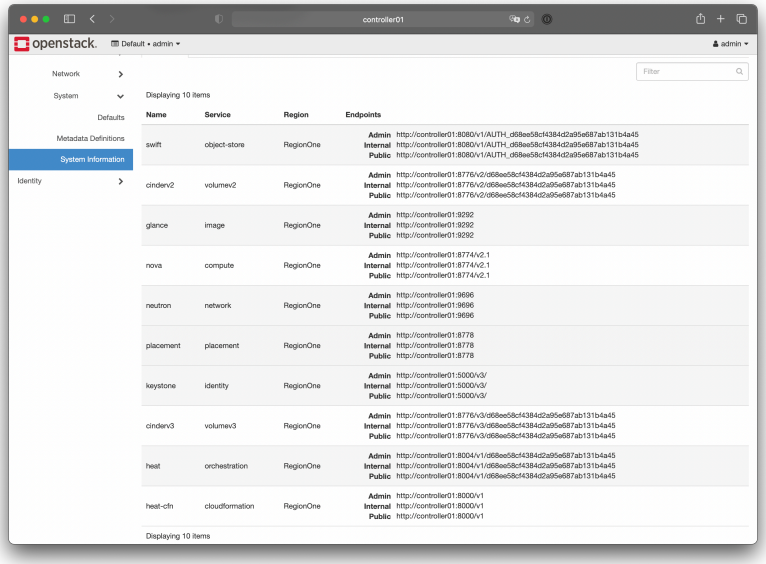


Figure 16: Heat: Orchestration Services.

# 5 Conclusion

In conclusion, we were able to get most services running on our OpenStack deployment with the lone exception being Swift. We can certainly appreciate the learning experience of manually deploying OpenStack as it not only gave us insight into how to install the various services, but also how to troubleshoot. There were many times where we needed to go back and delete a service, user, group, etc. in order to retry an installation. This homework, while challenging, was very beneficial in learning an overview of OpenStack. Figures 17 and 18, page 34 show our Horizon dashboard with all services appearing along with their endpoints.

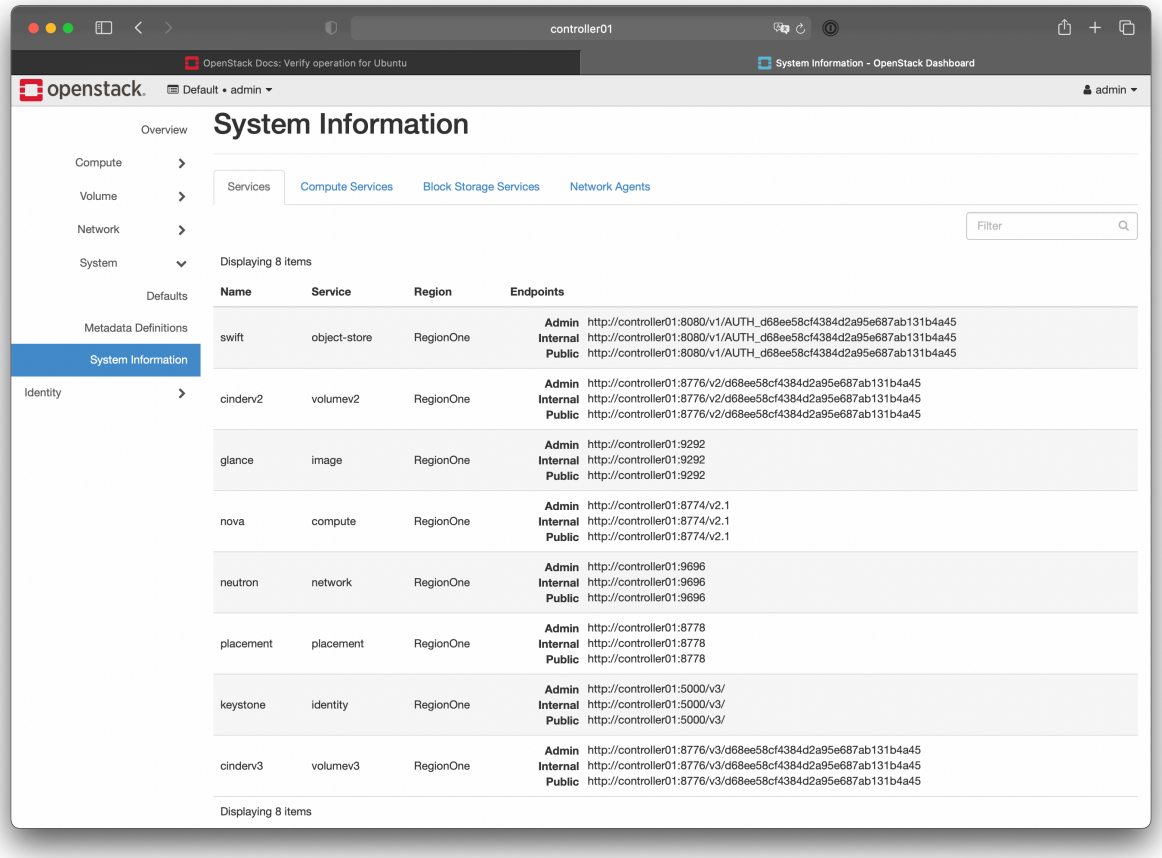


Figure 17: Conclusion: Horizon Dashboard Showing All Services.

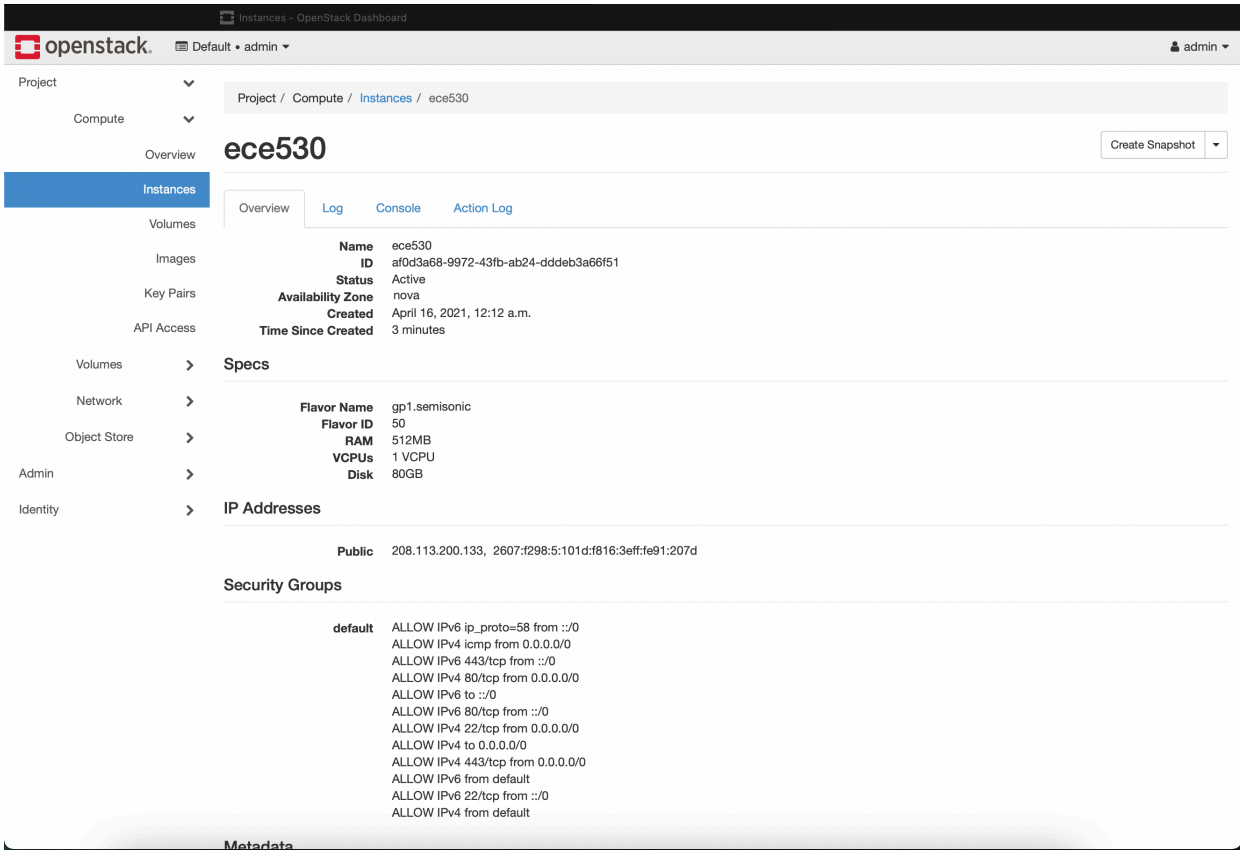


Figure 18: Conclusion: Horizon Dashboard Showing Created VM.

# 6 References

Denton, James *Learning OpenStack Networking - Third Edition*, Packt Publishing, 2018. O'reilly Media, https://learning.oreilly.com/library/view/learning-openstack-networking/9781788392495/.

"Overview." OpenStack Docs: Overview, https://docs.openstack.org/install-guide/overview.html.